Universitat Autònoma de Barcelona

escola d'enginyeria

DISRUPTION MANAGEMENT –

THE AIRCRAFT RECOVERY PROBLEM

Memòria del Projecte Fi de Grau
De Gestió Aeronàutica
realitzat per
Pol Arias Melià
i dirigit per
Daniel Guimarans Serrano
Sabadell,12 de Juliol del 2012

Contents

Chapter 1

Introduction

The main purpose is to create a small program capable of solving the *Aircraft Recovery Problem*, that is included in the *Disruption Management* field.

In **Chapter 1** we will introduce all the concepts in relation with Disruption Management, the viability plan and the objectives. In the **second chapter**, the introduction to the Aircraft Recovery Problem and the state-of-art. In **Chapter 3**, Constraint Programming (CP) and the platform Eclipse will be presented. The **fourth chapter**, we will introduce the CP formulation of the problem and some implementation issues. In **Chapter 5** two scenarios will be tested and both results will be given. Finally, in **Chapter 6** some conclusions and further work will be given to complete the project.

1.1 Planification and Viability

The project was carried out according to a schedule. The schedule was the first thing done and was planned using a Gantt.

In Table 1.1, are shown the cost related to the project.

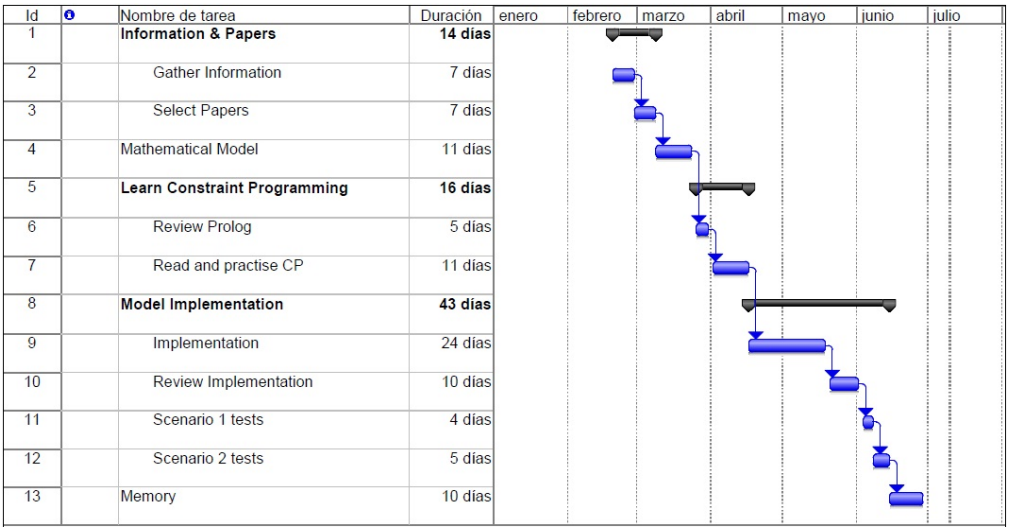

Figure 1.1: Gantt's project

Spent Time	6 months	0 euros
Computer	Department's Computer	0 euros
Software	ECLiPSe (Open Source)	0 euros
Total		0 euros

Table 1.1: Project's attached costs

Basically, the project is feasible as does not have any costs attach. As being a degree's final project the hours dedicated to it are chargeable, the computer used was a Toshiba that belong to the university and the software was an open source program.

1.2 Introduction to Disruption Management in Airline Industry

Operational disruptions are defined as a deviation from originally planned operations. The airline industry is notably one of the most affected industries regarding operational disruptions and optimization based methods and tools are commonly used in planning, disruptions, and scheduling. Plans are usually made several months prior to the actual day of operation. As a consequence, changes often occur in the period from the construction of the plan to the day of operation. Thus, optimization tools play an important role also in handling these changes.

Some examples of the importance of optimization in the airline industry are shown on figures 1.2, 1.3, 1.4 . Figure 1.2 shows the percentage of delayed flights. It can be observed that some years the delayed flights represent up to 25 % of the total. Consequently, the complementary graphic can be seen in Figure 1.3. Finally, Figure 1.4, shows the percentage of cancelled flights.

This project is focused on the disruption reallocation. Either if the disruptions occur by external factors or as the result of human action, there are three major research areas on disruption reallocation:

- *Aircraft Reallocation*: When disruptions occur, the aircrafts must be reallocated to the remaining flights.
- *Crew Reallocation*: Likewise aircrafts but scheduling constraints must be considered as they have some limitations about the work hours.
- *Passengers Reallocation*: In this case, the main focus is to reallocate all the passengers to flights that with the same destination.

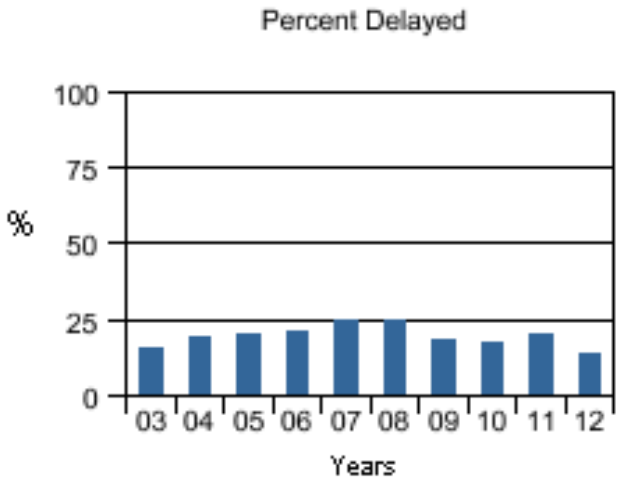

Figure 1.2: Percentage Delayed Flights

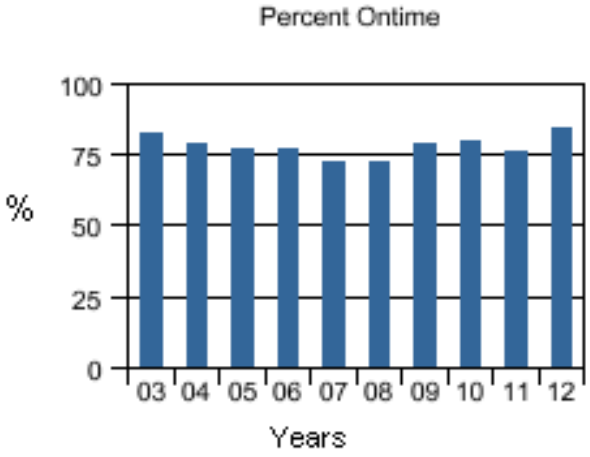

Figure 1.3: Percentage On Time Flights

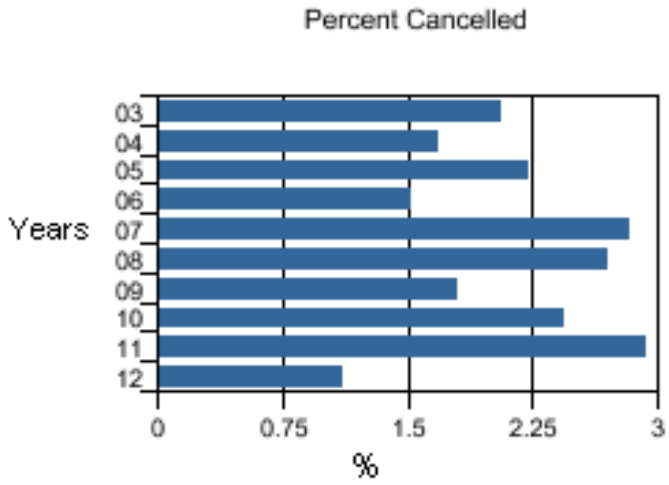

Figure 1.4: Percentage Cancelled Flights

However, at the day of operation, no planning tool have been able to cope with the complexity of the numerous constraints should be considered by unifying aircrafts, crew, and passenger concurrently in a single system.

1.3 Objectives

Like in all projects, in this one some objectives must be completed. There are two kind of objectives the main objective and the secondary or specific objectives. In this section, we will go through all of them.

1.3.1 Main Objective

The main objective of this project is to model the ARP from a constraint programming (CP) point of view . The information required for this project is extracted from previous papers that cope with the problem using heuristics,

metaheuristics or using network-models.

Also, two scenarios will be tested to verify that the implementation is correct. In a further chapter both scenarios will be explained and tested.

1.3.2 Specific Objectives

In order to complete de main objective, a more specific targets must be achieved. These milestones are important as they are previous steps to accomplish the implemented and final model.

- Familiarize with *Constraint Programming* (CP)
- Formally define all the constraints that will be used in the model. The constraints have to be concise and well formulated.
- Complete the mathematical model which will be used as the basis of the implementation.

Chapter 2

Aircraft Recovery Problem

The Aircraft Recovery Problem (ARP) is well defined in [9] and arises when unforeseen events have disrupted an existing flight schedule, causing a "cascade" propagation affecting the successors flights e.g. bad weather on an airport causes flights to be delayed. The first priority then is to restore the flight schedule as much as possible, using the existing aircrafts, i.e. minimize the number of cancellations and the total delay. ARP has been given other names by different researchers and its precise definition varies accordingly. Hence, it is important to define ARP as it is understood on [reference]: "Given an original flight schedule and one or more disruptions, the Aircraft Recovery Problem consists of delaying flights, canceling flights, and swapping aircraft to flight assignments in order to create a feasible and more preferable revised flight schedule. The term swap denotes that two flights, designated to be undertaken by two specific aircrafts, are interchanged between these aircrafts".

The flight schedule includes all flights flown within a certain period of time by a given fleet including the original departure, the expected flight durations, and the connections between airports.

The ARP is included in the non-deterministic polynomial-time hard (NP-Hard) problems. NP stands for non-deterministic polynomial-time, which means that a problem can be solved in a polynomial time. So NP-Hard, means that the bigger the problem is the computational time needed to solve it increases exponentially.

2.1 State-of-the-Art

By far most of the work on operational recovery problems has been reported on the aircraft resource. Since crews can be repositioned fairly easy and standby crews are often available, the aircraft are seen as the scarce resource. One of the firsts publications about aircraft recovery problem was published by Teodorovic and Guberinic [11] . In their paper, one or more aircraft are unavailable and the objective is to minimize the total passenger delays by reassigning flights and retiming flights. The formulation is based on a network-model. A branch-and-bound method is used to solve the problem.

In a later work the same authors considered aircraft shortage and proposed an improved approach [12] . A heuristic algorithm based on dynamic programming was developed to solve this lexicographic optimization problem . The constructed model allows cancellations, retimings and swaps. The main objective is to minimize the number of cancellations.

Love et al.[9] presented a paper for the aircraft recovery problem based on local search. The schedule is represented by the lines of work for each available aircraft. In order to resolve the model, cancellations, delays or reassignment of aircrafts are considered. Reassignments, both within a single fleet or between fleets, can be handled. The objective function is to minimize the recovery costs. Costs are related delays, cancellations and swaps. It is

even possible to assign costs on the individual flights in the recovery period, in order to weight the importance of the different flights. This approach was tested on the short-haul operation of British Airways (79 aircraft, 44 airports, 339 flights) and achieved promising solutions within 10 seconds.

Arguello et al. [4] [8] presented a method based on the metaheuristic Greedy Randomized Adaptive Search Procedure (GRASP) to reschedule the aircraft routings if one or more aircrafts are unavailable. The heuristic is capable of canceling and retiming. As in the approach of Love et al. [9] also allows swaps between fleets. The goal is to produce a recovery schedule in order to restore the original schedule. The cost to be minimized includes measures of passenger inconvenience and lost flight revenue. The approach was also tested in a fleet of 16 aircraft, 42 flights and 13 airports.

In the papers of Yan and Yang [14] and Yan and Tu [15] four models were developed to deal with the temporary unavailability of one single aircraft. The models are developed specifically for small airlines. In the first model, it is possible to cancel flights in order to repair the schedule. The second model has an increased complexity and allows for both the cancellation of flights and ferrying of spare aircraft. The third model considers cancellation and retiming, and the last model incorporates all of the possible decisions, and adds ferrying of aircraft. In all models swaps are allowed within a fleet. The objective in all four models is to minimize the cost of schedule repair, which includes passenger revenue. The first two models are rather simple and are built as network flow models, which make them possible to be solved to optimality very fast. The other two models contain side constraints, making them hard to solve. They are solved using Lagrangian relaxation [5] and subgradient optimization [7]. In all models, neither crew nor maintenance is considered. In Yan and Tu [15], a multi-fleet version of the model described

above is presented. In this case, a larger aircraft type can be assigned to a flight that originally was planned to be serviced with a smaller aircraft type. Furthermore, flights that have more than one stop, some or all stops can be deleted. The case study presented in [14] is based on China Airlines data. The fleet consists of 12 aircraft serving 15 cities and covering 319 flights a week. First two models solve the instances to optimality while the next ones get within 1 percentage of optimality in a matter if minutes. In [15] the experiments are performed on a test set of 26 aircraft divided into 3 different fleets performing 273 flights a week. In total, the developed methods were tested with 534 different scenarios. All problems were solved to optimality or at most 1 % from optimality within 5.5 minutes.

Chapter 3

Introduction to Constraint Programming

Constraint programming (CP) is a programming paradigm that uses constraints to relate variables to one another. It differs from other programming languages, as it is not necessary to specify a secuence of steps to execute, but rather the properties. The main applications areas to date are scheduling, routing, planning, configuration, etc. Models in CP are based in three steps: creating the *variables*, associate their corresponding *doamains*, and finally the base of CP, to declare all the *constraints* relating all the variables. As it is not needed the implementation of a *search* procedure, it gives more flexibility to the model, and usabilty to the user. In fact, if new variable or constraints wanted to be improved , it is not needed to do major changes in the structure, as is not implemented the search to solve the problem.

The practical benefits of CP really began to emerge when it was embedded in a programming language. Thus, CP is usually found embedded in a logic programming language, such as *Prolog*. In that case, it is called *Constraint Logic Programming* (CLP), but it does not necessarily mean that CP

is restricted to CLP. Constraints can be integrated also to typical imperative languages like C/C++, e.g. COMET [13] or ILOG [1] , and Java, e.g. Cream [10]. The program implementing the methodologies presented in this project have been made using the CLP platform ECLiPSe [3].

CLP combines logic, that is used to specify a set of possibilities to be explored by simple inbuilt search methods, with constraints, which are used to minimize the search, by eliminating some impossible solutions or by narrowing down the set of possibilities.

As explained before, CP solves problems based on constraints, in order to find a feasible solution satisfying all the constraints. This class of problem is known as *Constraint Satisfaction Problem* (CSP) and the maine mechanism of solving it, is *constraint propagation*.

Constraint propagation works by reducing domains of variables, strengthening constraints, or creating new ones. This leads to a reduction of the search space, making the problem easier to solve by some algorithms. Basically, with constraint propagation it shows how a decision affects the result.

A solution to a CSP is a full assignment to the variables of the problem, in such a way that all constraints are satisfied at once. We may want to find:

- just one solution, with no preference to which one,
- all solutions,
- an optimal, or at least a good solution, given some objective function defined in terms of some or all of the variables. In this case, the CSP becomes a Constraint Optimization Problem (COP).

Finally, an important contribution of CP is to allow the end user to control the search. The topic of search comes from the heart of AI, which

has developed several algorithms to perform the search in a solution space. End users search control is achieved by combining generic techniques, when the generation of the whole search tree is unfeasible, and problem-specific techniques, when there is an extra knowledge about special features of the problem. Thus, while mathematical programming is mainly based in the application of certain algorithms to a model, CP allows the user to take some decisions on the search stage like the order of instantiation of the variables and the order of selection of values from domains. Depending on those decisions the way decisions are made is totally different and the performance of the search algorithm can be highly affected.

3.1 Constraint Satisfaction

As it has been said above, constraint satisfaction is related to problems defined over finite domains. Solutions to a CSP can be found by searching (systematically) through the possible assignments of values to variables, that is generating the whole search tree. Search methods can be divided into two broad classes: those that traverse the space of partial solutions (or partial value assignments), and those which explore the space of complete value assignments (to all variables) stochastically.

From the theoretical point of view, solving a CSP is trivial using systematic exploration of the solution space. But that is not true from the practical point of view, where the efficiency takes place. Even if systematic search methods (without additional improvements) look very simple and non-efficient, they are important because they make the foundations of more advanced and efficient algorithms.

The simplest algorithm that searches the space of complete labelings, is

called Generate-and-Test (GT). The idea of GT is very simple: firstly, a complete labeling of variables is randomly generated and, consequently, if this labeling satisfies all the constraints then the solution is already found; otherwise, another labeling is tried.

The GT algorithm is clearly a weak generic algorithm used only if everything else failed. Its efficiency is very poor for two reasons: it has a non-informed generator and there is a late discovery of inconsistence. There are two ways to improve efficiency in GT:

- To program a smart (informed) generator of valuations, i.e. able to generate the complete valuation in such a way that the conflict found by the test phase is minimized.

- To merge the generator and the tester, i.e. the validity of the constraint is tested as soon as its respective variables are instantiated. This method is used by the backtracking approach.

As said before, *Backtracking* (BT) is a method used for solving CSPs by incrementally extending a partial solution that specifies consistent values for some of the variables, towards a complete solution, by repeatedly choosing a value for another variable consistent with the values in the current partial solution.

BT is a merge of the generating and testing phases of GT. The variables are labeled sequentially and as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If a partial solution violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available. Clearly, whenever a partial instantiation violates a constraint, backtracking is able to eliminate a subspace from the Cartesian product of all variables do-

mains. Hence, backtracking is strictly better than GT. However, its running complexity for most non-trivial problems is still exponential.

There are three major drawbacks of the standard BT:

- *Thrashing*: it is a repeated failure (and consequent backtrack) due to the same reason. This happens because there is no information stored when a failure occurs. Thus, if there is a similar situation in the future the search will also fail and backtrack.

- Redundant work: conflicting values of variables are not remembered. This makes the search fail the same way in different branches of the tree.

- Late detection of conflicts: conflict is not detected before it really occurs.

More sofisticated and improved methods were proposed as *consistency techniques* to detect the inconsistency of partial solutions sooner. There exist several consistency techniques, but most of them are not complete. For this reason, these techniques are rarely used alone to solve a CSP completely. The names of basic consistency techniques are derived from the graph notions.A binary CSP can be represented as a constraint graph where nodes correspond to variables and edges are labeled by constraints . Although this representation can be applied only to binary CSPs, it is easy to show that every CSP can be transformed to an equivalent binary CSP [6]. However, in practice this operation is not likely to be worth doing and it is easier to extend the algorithms so they can tackle non binary CSPs as well.

Among consistency techniques, some of the most common are:

- *Node-Consistency*: it removes values from variables domains that are

inconsistent with constraints involving one variable, i.e. unary constraints. It is the simplest consistency technique.

- *Arc-Consistency*: it removes values from variables domains which are inconsistent with constraints involving two variables, i.e. binary constraints.

- *Path-Consistency*: it requires for every pair of values of two variables x and y satisfying the respective binary constraint that there exists a value for each variable along some path between x and y such that all binary constraints in the path are satisfied.

- *K-Consistency* and *Strong K-Consistency*: a constraint graph is k-consistent if for every system of values for $k-1$ variables satisfying all the constraints among these variables, there is a value for an arbitrary k^{th} variable such that the constraints among these variables are satisfied. A constraint graph is strongly k-consistent if it is j-consistent for all $j \leq k$. All previously mentioned techniques can be generated by k-consistency and strong k-consistency.

Attention should be paid to the use of these consistency techniques. They provide a good mechanism to remove inconsistent values from variables domains during search, but they often penalize with respect to efficiency terms. For this reason, they are often neglected on designing efficient search algorithms and substituted by heuristic approaches.

Finally, as said before, to cope with *Constraint Optimization Problems* (COPs), to find the optimal solution of a problem, it is needed to take into account a cost function. The appropriate search mechanism is a variation of BT, called *Branch-and-Bound* (BB). During the search, BB maintains the current best value of the cost function (*bound*) and, each time a solution with

a smaller cost is found, its value is updated. There are many variations on the BB algorithm. One consideration is what to do after a solution with a new best cost is found. The simplest approach is to restart the computation with the bound variable initialized to this new best cost. A less naive approach is to continue the search for better solutions without restarting. In this case, the cost function upper bound is constrained to the bound variable value. Each time a solution with a new best cost is found, this cost is dynamically imposed through this constraint. The constraint propagation triggered by this constraint leads to a pruning of the search tree by identifying the nodes under which no solution with a smaller cost can be present.

3.2 ECLiPSe

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. Eclipse uses both Java or Constraint Programming languages, in this project we will use the Constraint Programming language. It is a merge of three systems:

- One enabled complex problems to be solved on multiprocessor hardware, and eventually on a network of machines.
- The second supported advanced database techniques for intelligent processing in data-intensive applications.
- The third system was CHIP, see Aggoun et al. [2]. CHIP incorporated the concept of a constraint satisfaction problem into the logic programming paradigm by using constraint variables ranging over user defined finite domains. During the computation the values of the constraint

variables are not known, only their current domains. If a variable domain shrinks to one value, then that is the final value of the variable.

The first released interface to an external state-of-the-art linear and mixed integer programming package was in 1997. The integration of the finite domain solver and linear programming solver, supporting hybrid algorithms, came in 2000. In 2001 the ic library was released. It supports constraints on Booleans, integers and reals and meets the important demands of practical use: it is sound, scalable, robust and orthogonal.

Chapter 4

ARP Formulation and Implementation

It should be noticed that none publications have been found introducing a CP formulation about the ARP. As explained before CP formulation is based in three entities: 1)variables, 2)their domains and 3) the constraints relating these variables. Below, we detail each of these entities corresponding to our formulation of the ARP.

4.1 CP Formulation

The variables used in this formulation are:

- $X_{af} = X_{11}..X_{nm}$ is a variable used for the assignation of one airplane to one flight, with domain $X_{af} :: [0, 1]$. Defined over the sets A and F, is the first decision variable.
- $A = a_1..a_n$ are the set of airplanes available in each airport;

- $F = f_1..f_m$ where f is the number of flights available;

- $C_f = C_1..C_m$ is the set of flight cancelled, this variable is the second used in the objective function.

- $D_f = d_1..d_m$ are the durations of each flight;

- $ST_f = ST_1..ST_m$ are the scheduled time of departure of each flight;

- $Limit$ is the variable used to delimitate the sum of the delays, which can derive to a cancel;

- $Y_{fpp'}$ is a three dimensional list of lists used to know, of which airport each flight departs from, and which airport arrives to.

- $ID_{af} = id_{11}..id_{nm}$ are the set of delays introduced initially of each flight.

- $D_{af} = d_{11}..d_{nm}$ are the set of delays calculated by the program, with domain: $D_{af} :: [0..Limit]$

- $S_{af} = S_{11}..S_{nm}$ are the set of starting times, with domains: $S_{af} >= 0$ and $S_{af} =< ST_f + Limit$

The variable above introduced are related through a set of constraints that are described next:

$$\sum_{a \in A} X_{af} \leq 1 \; \forall f \in F \tag{4.1}$$

First, equation (4.1) forces a flight 'f' to be assigned to only one aircraft 'a' .Thus, one flight can only be assigned to one aircraft, but it allows one aircraft to have more than one flight. We have introduced two constraints to handle the flights cancelations:

$$X_{af} D_{af} \geq Limit \Leftrightarrow C_f = 1 \tag{4.2}$$

$$\sum_{a \in A} X_{af} = 0 \forall f \in Cf \tag{4.3}$$

The first equation (4.2) is used to delimit the delays, as it would not be realistic to have unlimited delays, variable. As mentioned, 'Limit' is the maximum time that a flight can be delayed. If the delay is greater than Limit, then the flight is cancelled, and so the variable C_f is set to 1. As for the second equation (4.3), if the flight is cancelled, no aircraft can be assigned and so the related variables X_{af} should be null.

To model the starting times of the flights according to the assigned aircraft, including the delays, we use the next equation:

$$S_{af'} \geq X_{af'} X_{af} (S_{af} + D_f) \tag{4.4}$$

Expression (4.4) uses the predecessor starting time plus the duration of the flight to calculate the starting time of the successor. As not all the flights have a successor (i.e the initial flights), and we need to take into account the delays that are defined in the scenario.

$$S_{af} \geq X_{af}(ST_f + D_{ay}) \tag{4.5}$$

Using equation (4.5), the problems with the initialization of the scenario are fixed as the starting time is forced to be greater or equal than the scheduled time plus the delays defined in the scenario.

To calculate the delay, the following constraint is defined:

$$D_{af} = (S_{af} - ST_f)X_{af} \tag{4.6}$$

In the equation above (4.6), the scheduled time and the starting time, previously bounded, are used to calculate the delay of each flight assigned to an aircraft.

The last constraint to complete the model is:

$$\sum_{f \in F} X_{af}Y_{fpp} \ \leq 1 \ \forall a \in A, \ \forall p, p' \in P \mid p \neq p' \tag{4.7}$$

Equation (4.7) is used to delimit how many flights an aircraft may have assigned in one airport. In this case the variable Y_{fpp} is used to know which flights depart from each airport.

Finally, the objective function is formed by the cumulative total delays and the number of cancelations:

$$\min \ \alpha \sum_{a \in A} \sum_{f \in F} X_{af}D_{af} + \beta \sum_{f \in F} C_f \tag{4.8}$$

The equation (4.8) uses the variables D_{af} and C_f to calculate the total delay and the flights canceled. As the cost of a cancelled flight is far superior than the cost of a limited delay, the parameters α and β have been introduced to adjust the weight of these two decision variables.

4.2 Implementation characteristics

As in every formulation, when it comes the implementation part, we deal with some difficulties, in this section some illustrative examples of how delays, cancellations and swaps works will be introduced and some tricky implementation steps will be explained.

In Figure 4.1, shows how a swap will be produced between two flights and two aircrafts. Figure 4.2 displays how a delayed is handled. Finally, in Figure 4.3 shows how a flight is cancelled.

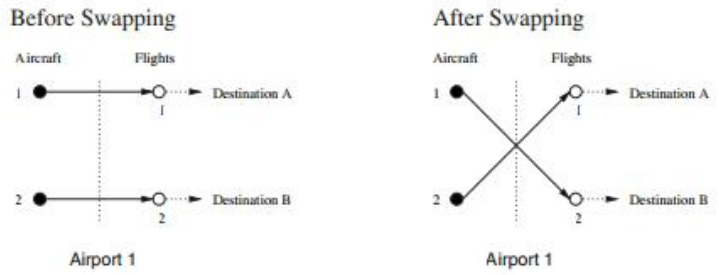

Figure 4.1: Swap Flights

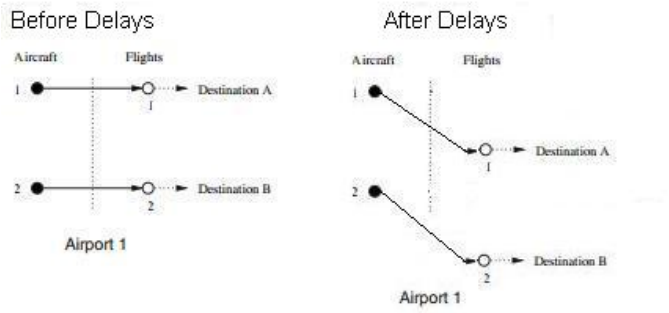

Figure 4.2: Delayed Flights

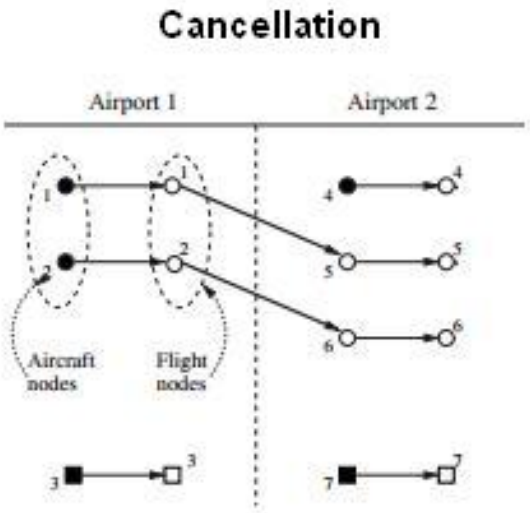

Figure 4.3: Cancelled Flights

Regarding the search, it has been implemented using the branch and bound algorithm (BB). In ECLiPSe the BB is already implemented, also it gives us the opportunity to use other search methods such as the *Limited Discrepancy Search* (LDS), *Depth Bounded Search* (DBS), etc.

Chapter 5

Application and Results

The ARP has been tested in two different scenarios and we have had some results, but as none CP implementation has been found, the comparison between this approach is impossible, as other papers use heuristics and these are more efficient.

5.1 ARP scenarios

The first scenario was used to test a small problem of the ARP, in this case, were used 3 aircrafts, 7 flights, and 4 airports. As shown in Figure 5.1 the first two aircrafts are allocated on the first airport and the third one on the second. As it can be seen any flight goes from one airport to another leaving one in the middle.

Some inputs are introduced in Table 5.1 to complete the scenario, it shows which flights have delays, and the depart scheduled time.

On Table 5.1 are shown all the scheduled times needed to run the first Scenario, also to test an scenario with disruptions are introduced some delays on Airport 1, where Flight 1 and Flight 2 depart.

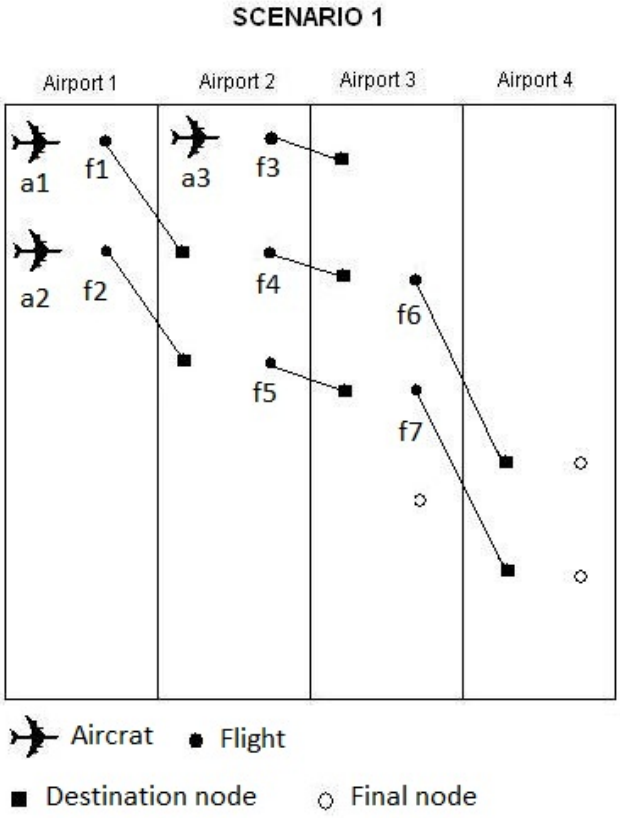

Figure 5.1: Scenario 1

On Figure 5.2 is done with 5 aircrafts, 17 flights, and 7 airports. In this case, some flights jump from one airport to another, leaving one in the middle. Here, two aircrafts are allocated on the first airport, two more aircrafts on the third airport and finally the fifth aircraft departs from airport 4.

Also, in this scenario some inputs are introduced, Table 5.2. In this case, airport 3 is where we include the disruptions, delaying every flights 120 minutes.

Flights	Scheduled Time	Delay
Flight 1	0	60
Flight 2	20	70
Flight 3	0	0
Flight 4	130	0
Flight 5	150	0
Flight 6	100	0
Flight 7	250	0

Table 5.1: Scheduled and delay times for Scenario 1

Flights	Scheduled Time	Delay
Flight 1	0	0
Flight 2	20	0
Flight 3	100	0
Flight 4	130	0
Flight 5	0	120
Flight 6	30	120
Flight 7	210	120
Flight 8	250	120
Flight 9	0	0
Flight 10	330	0
Flight 11	150	0
Flight 12	250	0
Flight 13	420	0
Flight 14	470	0
Flight 15	240	0
Flight 16	250	0
Flight 17	350	0

Table 5.2: Scheduled and delay times for Scenario 2

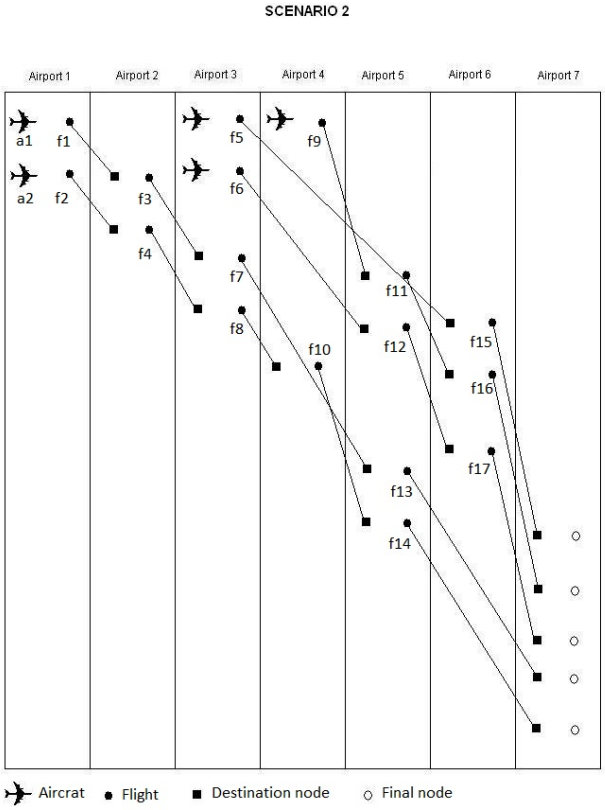

Figure 5.2: Scenario 2

5.2 Results

The tests were conducted using a 2Gb RAM and Core Duo of 2.6 GHz. In both scenario were conducted two tests, the first one, without using delays and the second applying some delays as explained before. The firs test is useful as it shows how the problem is solved and letting us look for the initial solution. Both scenario were tested using an $\alpha = 10$ and a $\beta = 1000$. The β is this big as we do not want any flight canceled.

In Figure 5.3, the allocation of which flight is assigned to which aircraft without any delay. The "cascade" propagation can bee seen here in a clear

way.

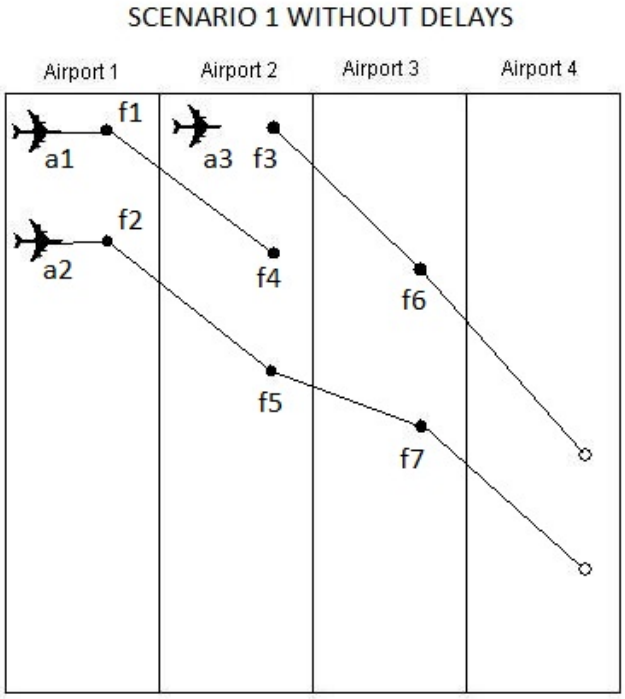

Figure 5.3: Scenario 1 without delays

On the other hand, when the delays are applied, some changes appear. In Figure 5.4, we observe how aircraft 1 takes flight 5 instead of flight 4, and then takes flight number 7. The final cost in this case is 4380.

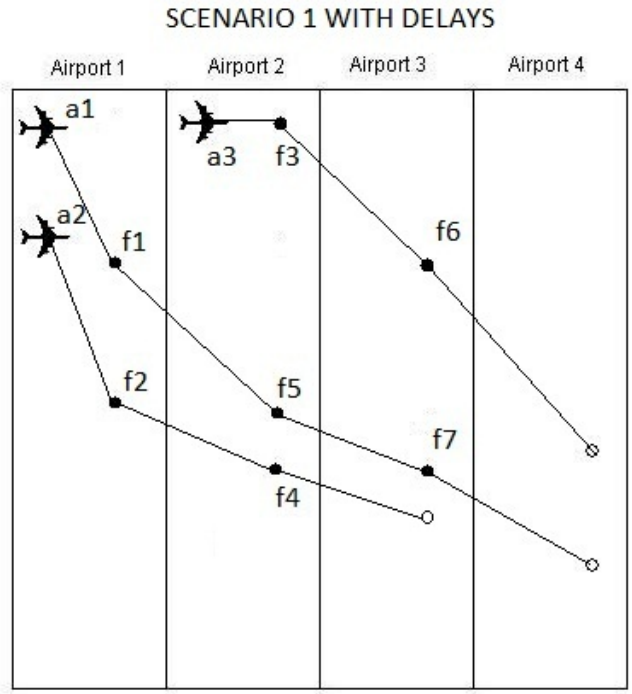

Figure 5.4: Scenario 1 with delays

In figure 5.5, scenario 2 is tested without delays, and the result as in scenario 1 shows how the initial solution is.

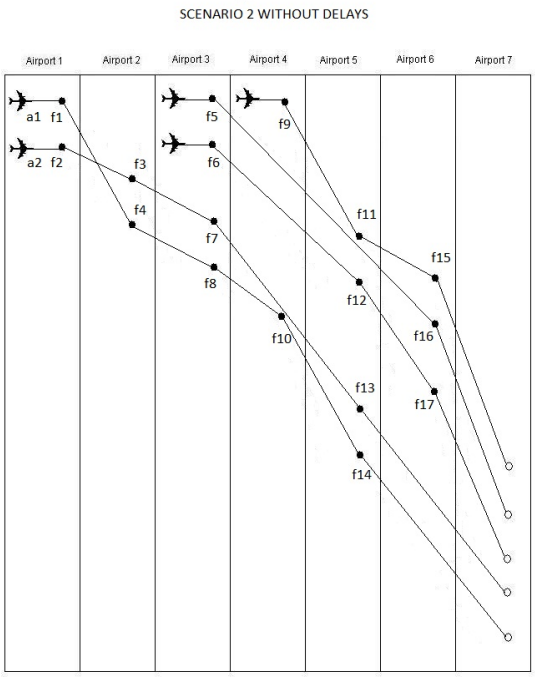

Figure 5.5: Scenario 2 without delays

In a more complex scenario changes are more noticeable. In Figure 5.6, flights 5, 6, 10, 12 , 13, 14, 16, 17 are assigned to a different aircraft. As a result of this changes the final costs is up to 14800.

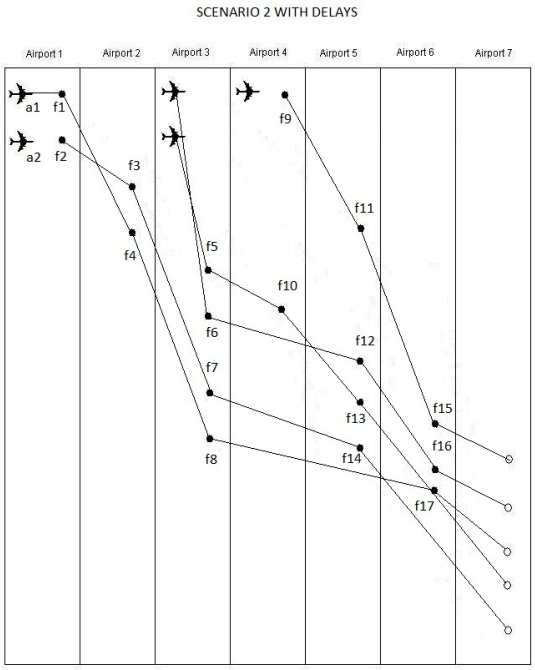

Figure 5.6: Scenario 2 with delays

Chapter 6

Conclusions

The main objective of this project was to model the ARP in constraint programming. As seen during the project we demonstrate that the objectives are achieved, and that the ARP can be solved using CP. Our methodology allows more flexibility, as small changes are needed to include new restrictions to the problem.

Our project contributes to a new line of research of the ARP, using the CP approach. We think that this approach could lead to a new perspective for tackling this problem as the final goal is to include passengers and crew reallocation. Using CP, the more constraints we have, the better.

But as every methodology it has its limitations. The CP search can not compete with heuristics and metaheursitcs, as it searches all the possible combinations finally finding the optimal solution, and thus a large computational time is required. This is a handicap when a big scenario is tested.

From a personal point of view, the project has been hard but interesting, as it has allowed me to introduce myself into the optimization world, more specifically into constraint programming, which I have really enjoyed.

6.1 Future Work

Further research should be focused on the implementation of passengers and crew reallocation jointly with the ARP, which is the final goal. Furthermore, from a more specific point of view, the search should be improved, as the branch and bound is not an efficient algorithm.

Bibliography

[1] *ILOG Solver 6.3 Reference Manual*, 2006.

[2] A. Aggoun, M. Dincbas, A. Herold, H. Simonis, and P. Van Hentenryck. The chip system. *Technical Report*, 1987.

[3] K. Apt and M.Wallace. Constraint logic programming using eclipse. *Cambridge University Press*, 2007.

[4] Michael F. Arguello, Jonathan F. Bard, and Gang Yu. Models and methods for managing airline irregular operations. *Operations Research in the Airline Industry*, pages 1 – 45, 1997.

[5] M. Fisher. The lagrangean relaxation method for solving integer programming problems. *Management Science*, 27:1–28, 1981.

[6] F.Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. *Technical report ACT-AI-222-89*, 1989.

[7] M. Held and R.M. Karp. The travelling salesman problem and minimum spanning trees: part ii. *Mathematical Programming*, 1:6–25, 1971.

[8] Michael F. Arguello, Jonathan F. Bard and Gang Yu. A grasp for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization*, 5:211 – 228, 1997.

[9] Michael Løve, Kim Riis Sørensen, Jesper Larsen and Jens Clausen. Disruption management for an airline -rescheduling of aircraft. *Applications of Evolutionary Computing*, 2279:315 – 324, 2002.

[10] N. Tamura. Cream version 1.2 programmers guide. *Available online: http://bach.istc.kobe-u.ac.jp/cream/*, 2004.

[11] Dusan Teodorovic and Slobodan Guberinic. Optimal dispatching strategy on an airline network after a schedule perturbation. *European Journal of Operational Research*, 15:178 – 182, 1984.

[12] Dusan Teodorovic and Goran Stojkovic. Model for operational daily airline scheduling. *Transportation Planning and Technology*, 14:273 – 285, 1990.

[13] P. van Henteryck and L. Michel. Constraint-based local serach. *The MIT Press*, 2009.

[14] S. Yan and D.-H. Yang. A decision support framework for handling schedule perturbation. *Transportation Research B*, 30(6):405 – 419, 1996.

[15] Shangyao Yan and Yu-Ping Tu. Multifleet routing and multistop flight scheduling for schedule perturbation. *European Journal of Operational Research*, 103:155 – 169, 1997.